Expressões regulares: introdução

Idéia básica

Uma expressão regular é uma notação para representar **padrões** em strings. Serve para validar entradas de dados ou fazer busca e extração de informações em textos.

Por exemplo, para verificar se um dado fornecido é um número de 0,00 a 9,99 pode-se usar a expressão regular `\d,\d\d`, pois o símbolo `\d` é um curinga que casa com um dígito.

O verbo *casar* aqui está sendo usado tradução para *match*, no sentido de *combinar*, *encaixar*, *parear*. Dizemos que a expressão `\d,\d\d` casa com 1,23 mas não casa com `123` (falta a vírgula) nem com `1,2c` ("c" não casa com `\d`, porque não é um dígito).

O termo em inglês é *regular expression* de onde vem as abreviações *regex* e `re` (o nome do módulo Python). Na ciência da computação, o termo tem um significado bem específico (veja *expressão regular* no *Glossário*).

Algums exemplos

Veja alguns exemplos com breves explicações para ter uma idéia geral:

`\d{5}-\d{3}`

O padrão de um CEP como 05432-001: 5 dígitos, um `-` (hífen) e mais 3 dígitos. A sequência `\d` é um *metacaractere*, um curinga que casa com um dígito (0 a 9). A sequência `{5}` é um *quantificador*: indica que o padrão precedente deve ser repetido 5 vezes, portanto `\d{5}` é o mesmo que `\d\d\d\d\d`.

`[012]\d:[0-5]\d`

Semelhante ao formato de horas e minutos, como 03:10 ou 23:59. A sequência entre colchetes `[012]` define um *conjunto*. Neste caso, o conjunto especifica que primeiro caractere deve ser 0, 1 ou 2. Dentro dos `[]` o hífen indica uma faixa de caracteres, ou seja, `[0-5]` é uma forma abreviada para o conjunto `[012345]`; o conjunto que representa todos os dígitos, `[0-9]` é o mesmo que `\d`. Note que esta expressão regular também aceita o texto `29:00` que não é uma hora válida (horas válidas serão o tema de um dos *Exercícios*).

`[A-Z]{3}-\d{4}`

É o padrão de uma placa de automóvel no Brasil: três letras de `A` a `Z` é seguidas de um `-` (hífen) seguido de quatro dígitos, como `CKD-4592`.

Sobre os sinais «» usados neste texto

Ao descrever de modo genérico alguma parte da sintaxe das expressões regulares usamos neste documento os símbolos «», para indicar uma parte que deve ser fornecida pelo usuário.

Por exemplo, a referência a um grupo tem a sintaxe `\«n»` onde `«n»` é o número do grupo a ser recuperado. Os sinais «» não fazem parte da sintaxe, então a referência ao terceiro grupo escreve-se como `\3`.

De modo semelhante, a sintaxe de quantificador moderado é `«q»?`, onde `«q»` é qualquer quantificador, como `*` em `*?` ou `{1,3}` no caso de `{1,3}?`.

Metacaracteres

Um *metacaractere* é um caractere ou sequência de caracteres com significado especial em expressões regulares. Os metacaracteres podem ser categorizados conforme seu uso.

Especificadores

Especificam o conjunto de caracteres a casar em uma posição.

metacaractere	conhecido como	significado
`.`	curinga	qualquer caractere, exceto a quebra de linha `\n` (ver *flag_dotall*)
`[...]`	conjunto	qualquer caractere incluido no conjunto
`[^...]`	conjunto negado	qualquer caractere não incluido no conjunto
`\d`	dígito	o mesmo que `[0-9]`
`\D`	não-digíto	o mesmo que `[^0-9]`
`\s`	branco	espaço, quebra de linha, tabs etc.; o mesmo que `[ \t\n\r\f\v]`

`\S`	não-branco	o mesmo que `[^ \t\n\r\f\v]`
`\w`	alfanumérico	o mesmo que `[a-zA-Z0-9_]` (mas pode incluir caracteres Unicode; ver *flag_unicode*)
`\W`	não-alfanumérico	o complemento de `\w`
`\`	escape	anula o significado especial do metacaractere seguinte; por exemplo, `\.` representa apenas um ponto, e não o curinga

Quantificadores

Definem o número permitido repetições da expressão regular precedente.

metacaractere	significado
`{n}`	exatamente *n* ocorrências
`{n,m}`	no mínimo *n* ocorrências e no máximo *m*
`{n,}`	no mínimo *n* ocorrências
`{,n}`	no máximo *n* ocorrências
`?`	0 ou 1 ocorrência; o mesmo que `{,1}`
`+`	1 ou mais ocorrência; o mesmo que `{1,}`
`*`	0 ou mais ocorrência
`«q»?`	modera qualquer um dos quantificadores acima (ver *Gula × moderação*)

Veja o grupo de exercícios *1. Especificadores e quantificadores*.

Âncoras

Estabelecem posições de referência para o casamento do restante da regex. Note que estes metacaracteres não casam com caracteres no texto, mas sim com posições antes, depois ou entre os caracteres.

metacaractere	significado
`^`	início do texto, ou de uma linha com o flag `re.MULTILINE`

`\A`	início do texto
`$`	fim do texto, ou de uma linha com o flag `re.MULTILINE`; não captura o `\n` no fim do texto ou da linha
`\Z`	fim do texto
`\b`	posição de borda, logo antes do início de uma palavra, ou logo depois do seu término; o mesmo que a posição entre `\W` e `\w` ou vice-versa
`\B`	posição de não-borda

Veja o grupo de exercícios *2. Âncoras*.

Agrupamento

Definem ou grupos ou alternativas.

metacaractere	**significado**
`(...)`	define um *grupo*, para efeito de aplicação de quantificador, alternativa ou de posterir extração ou re-uso
`...|...`	alternativa; casa a regex à direita ou à esquerda
`\«n»`	recupera o texto casado no n-ésimo grupo

Gula × moderação

Por default, todos os quantificadores são gulosos: tentam casar a maior quantidade possível de caracteres.

Para entender o que isso significa, considere que desejamos capturar o nome do primeiro tag (h1) no fragmento de HTML abaixo:

```
>>> html = '<h1>Alan Turing: 100 anos</h1>'
```

Usando o quantificador guloso `+`, acabamos por capturar o elemento inteiro, e não apenas o tag:

```
>>> res = re.match('<.+>', html)
>>> res.group()
'<h1>Alan Turing: 100 anos</h1>'
```

O resultado acima ocorre porque o sinal `>` casa em duas posições no texto, e casando na segunda posição o curinga guloso `.+` captura mais caracteres.

Se usamos o quantificador moderado `+?`, a expressão `.+?` fica satisfeita em capturar apenas os caracteres até o primeiro casamento de `>`:

```
>>> res = re.match('<.+?>', html)
>>> res.group()
'<h1>'
```